

Vorlesungsskript WS 2000/2001

Programmierung

Eine Einführung für Informatiker

Gert Smolka



Fachrichtung Informatik
Universität des Saarlandes, Saarbrücken

Dieses Skript wurde von Prof. Gert Smolka begleitend zur einer Vorlesung an der Universität des Saarlandes im Wintersemester 2000 / 2001 geschrieben. Die Web-Seite dieser Vorlesung finden Sie unter

<http://www.ps.uni-sb.de/courses/prog-ws00/>

Dieses Werk ist urheberrechtlich geschützt. Die Urheberrechte liegen beim Autor Gert Smolka.

Inhaltsverzeichnis

Vorwort	9
1 Crashkurs	13
1.1 Ausdrücke und Deklarationen	13
1.2 Prozeduren	21
1.3 Rekursive Prozeduren und Auswertungsprotokolle	23
1.4 Syntax und Semantik	27
2 Mathematische Objekte	31
2.1 Grundbegriffe	31
2.2 Tupel	34
2.3 Relationen und Funktionen	35
2.4 Rekursive Definition von Mengen	38
2.5 Konstruktion der natürlichen Zahlen	40
3 Typen und Prozeduren	45
3.1 Typen und Typdisziplin	46
3.2 Prozeduren sind Werte	48
3.3 Kartesische und kaskadierte Prozeduren	50
3.4 Polymorphe Prozeduren	52
3.5 Typinferenz	56
3.6 Op-Ausdrücke	58

3.7	Typen und Gleichheit	58
3.8	Bezeichnerbindung	59
3.9	Prozeduren und Auswertungsprotokolle	62
4	Listen	67
4.1	Grundbegriffe	67
4.2	Listen in Standard ML	68
4.3	Length, Append, Reverse, Concat und Tabulate	70
4.4	Map, Filter, Exists und All	72
4.5	Faltungsprozeduren	76
4.6	Hd, Tl und Null	77
4.7	Sortieren von ganzzahligen Listen	79
4.8	Polymorphe Sortierprozeduren	83
4.9	Prozeduren mit mehreren Regeln	84
4.10	Laufzeit von Prozeduren	86
5	Graphen und Bäume	91
5.1	Induktionsbeweise	91
5.2	Graphen	95
5.3	Bäume	99
5.4	Größenverhältnisse in Bäumen	101
5.5	Geordnete Bäume	103
5.6	Terme	106
6	Konstruktortypen und Ausnahmen	113
6.1	Varianten und Konstruktoren	113
6.2	Enumerationstypen	115
6.3	Typsynonyme	117
6.4	Case-Ausdrücke und abgeleitete Formen	117

6.5	Ein Typkonstruktor für Binärbäume	118
6.6	Prozeduren für Binärbäume	119
6.7	Ein Typkonstruktor für Terme	122
6.8	Ausnahmen	123
6.9	Optionen	126
7	Korrektheit und Laufzeit	135
7.1	Prozeduren und Funktionen	136
7.2	Rekursionsbäume	141
7.3	Terminierungsbeweise	142
7.4	Korrektheitsbeweise	145
7.5	Listeninduktion	147
7.6	Laufzeiten	149
7.7	Die Funktionsklassen O und Theta	153
7.8	Asymptotische Laufzeiten	154
7.9	Schnelle Exponentiation	157
7.10	Größte gemeinsame Teiler	158
7.11	Iterative Rekursion	160
7.12	Einführung von Akkumulatorargumenten	162
7.13	Höherstufige Prozeduren	166
7.14	Laufzeiten der Sortierprozeduren	167
7.15	Zusammenfassung	171
8	Direkt ausführbare Programme	177
8.1	Zeichen und Zeichenreihen	177
8.2	Übersetzung	181
8.3	Dateien und Prozesse	184
8.4	Lesen und Schreiben von Dateien	184

8.5	Interaktive Programme	186
8.6	Imperative Prozeduren	187
9	Typabstraktion und Modularisierung	195
9.1	Strukturen	196
9.2	Typabstraktion	198
9.3	Schlangen	202
9.4	Funktoren	205
9.5	Programmiersprachliche Details	206
9.6	Komponenten	208
10	Fallstudien	215
10.1	Binäre Suche	216
10.1.1	Suchbäume	216
10.1.2	Rot-Schwarz-Bäume	218
10.2	Große Zahlen	223
10.2.1	Zahldarstellungen	223
10.2.2	Additionsalgorithmus	225
10.2.3	Realisierung von großen Zahlen	226
10.3	Das Damenproblem	228
11	Semantik	237
11.1	Abstrakte Syntax	238
11.2	Statische Semantik	242
11.3	Dynamische Semantik	244
11.4	Einschritt-Semantik	249
11.5	Let-Ausdrücke und homogene Paare	254
11.6	Rekursive Prozeduren	255

12 Syntax	261
12.1 Lexikalische Syntax	261
12.2 Kontextfreie Syntax für Typen	266
12.3 Kontextfreie Syntax für arithmetischen Ausdrücke	271
12.4 Kontextfreie Syntax für F	274
12.5 Bemerkungen zur konkreten Syntax von Standard ML	279
13 Imperative Objekte	287
13.1 Zustände und Referenzen	287
13.2 Zähler	290
13.3 Funktionale und imperative Objekte	292
13.4 Stapel	293
13.5 Reihungen	295
13.6 Imperative Listen	298
13.7 Schleifen	301
13.8 Referenzen und ambige Deklarationen	303
13.9 Semantik von F mit Referenzen	304
14 Virtuelle Maschinen und Übersetzer	313
14.1 Eine virtuelle Maschine	314
14.2 Übersetzung einer imperativen Sprache	320
14.3 Statische Prozeduren	323
14.4 Realisierung der Prozedurbefehle	327
14.5 Endaufrufe	331
14.6 Übersetzung einer funktionalen Sprache	333
14.7 Die Software-Hardware-Schnittstelle	335
14.8 Realisierung von Programmiersystemen	339

15 Speicherverwaltung	345
15.1 Verzeigerte Blockdarstellung	345
15.2 Eine virtuelle Maschine mit Halde	348
15.3 Realisierung der virtuellen Maschine	350
15.4 Dynamische Prozeduren	354
15.5 Effiziente Realisierung kaskadierter Prozeduren	354
 Literaturverzeichnis	 357

Vorwort

Diese Vorlesung ist eine Einführung in die Programmierung. Sie richtet sich an Informatikstudenten im ersten Semester. Sie behandelt einerseits Datenstrukturen, Algorithmen, Typen, Typabstraktion und Module. Andererseits führt sie in die Theorie und Implementierung von Programmiersprachen ein.

Die Themen der Vorlesung sind vielfältig und gehören zu verschiedenen Teildisziplinen der Informatik. Die Vorlesung ist aber so konzipiert, dass sich ihre Teile schnell zu einem eleganten und in sich schlüssigen Gebäude zusammenfügen, in dem jeder Baustein seinen festen Platz hat.

Als Programmiersprache verwenden wir Standard ML, das auf elegante Weise grundlegende und fortgeschrittene Konzepte der programmiersprachlichen Informatik in ein praktisches Werkzeug umsetzt. Mit Standard ML verschwenden wir keine Zeit für programmiersprachliche Tricks und Ungereimtheiten. Wir legen keinen Wert darauf, alle Details der Programmiersprache Standard ML zu erklären.¹

Wir beginnen mit mathematischen Datenstrukturen und konzentrieren uns auf die rekursive Definition von Prozeduren (Funktionen) und Typen (Mengen). Standard ML liefert uns eine Sprache, in der wir diese Definitionen präzise aufschreiben können. Statt Papier verwenden wir zum Aufschreiben einen Computer, mit dem wir unsere mathematischen Programme dann auf Konsistenz überprüfen und nach Belieben ausführen können.

Wir lernen, Eigenschaften unserer Programme mit Hilfe von Induktion zu beweisen. Besonders interessieren wir uns dabei für Aussagen über die Korrektheit und die Laufzeit (schnell oder langsam) von Programmen.

Der Weg von mathematischen Programmen zu richtigen Programmen, die sich in die Software-Umgebung eines Computers nahtlos einfügen, ist kurz. Wir benötigen einen Übersetzer sowie Ein- und Ausgabeoperationen für Zeichenreihen.

¹ Wenn Sie mehr über Standard ML wissen wollen, können Sie mit dem Selbststudium eines Lehrbuches beginnen, dessen Hauptziel die Erklärung der Programmiersprache Standard ML ist. Wir empfehlen Ihnen das Lehrbuch von Ullman [20].

Als nächstes betrachten wir Datenstrukturen, deren Zustand sich mit dem Fortschreiten einer Berechnung ändert. Unter anderem versetzen uns solche Datenstrukturen in die Lage, abstrakte Speicher zu programmieren.

Wir beginnen jetzt, größere Programme zu schreiben. Dabei ist es in der Praxis essenziell, Abstraktionen einzuführen und deren konsistente Verwendung mit Hilfe von Typen automatisch zu überprüfen. Gleichzeitig beginnen wir, Programme in klar abgegrenzte Module zu zerlegen, die unabhängig von einander realisiert werden können. Mit Standard ML haben wir eine Programmiersprache, die Typabstraktion und Modularisierung besonders gut unterstützt.

Wir beherrschen jetzt eine Programmiersprache sowie einige Programmiertechniken und können damit nützliche Programme schreiben. Unsere Situation ist vergleichbar mit der eines Kindes, das seine Muttersprache leidlich beherrscht, dem aber die Regeln der Grammatik noch völlig fremd sind. Entsprechend analysieren wir jetzt die Struktur von Teilsprachen von Standard ML, die exemplarisch für alle Programmiersprachen sind. Wir lernen die Unterscheidungen zwischen konkreter und abstrakter Syntax sowie statischer und dynamischer Semantik kennen. Die im ersten Teil der Vorlesung behandelten mathematischen Datenstrukturen und rekursiven Definitionstechniken versetzen uns in die Lage, Syntax und Semantik von Programmiersprachen präzise zu definieren. Der Kreis beginnt sich zu schließen.

Wir haben jetzt die Voraussetzungen, um grundlegende Techniken für die Realisierung von Programmiersprachen auf Computern zu lernen. Diese erfolgt mit Programmen. Wir lernen, wie man einen einfachen Interpreter schreibt. Danach betrachten wir eine virtuelle Maschine mit automatischer Speicherverwaltung, die eine hardwarenahe Programmiersprache realisiert. Die Programme für die virtuelle Maschine erzeugen wir mit einem Übersetzer für eine Teilsprache von Standard ML. Den Interpreter, die virtuelle Maschine und den Übersetzer schreiben wir in Standard ML. Der Kreis ist geschlossen.

Danksagung Dieses Skript entstand begleitend zu einer Vorlesung, die ich an der Universität des Saarlandes bisher dreimal gehalten habe (WS 98/99, WS 99/00, WS 00/01).

Zuerst möchte ich Andreas Rossberg und Thorsten Brunklaus danken, die die Vorlesung als Assistenten betreut haben. Sie haben durch Diskussionen und Korrekturen beständig zu diesem Skript beigetragen. Weiter geht mein Dank an Marco Kuhlmann, der mir bei der typografischen Erstellung des Skripts geholfen hat und sich dabei schnell zum $\text{\LaTeX}$ -Experten entwickelt hat ($\text{\LaTeX}$ ist die Dokumentenbeschreibungssprache, in der das Skript geschrieben ist). Schließlich danke ich

den Übungsgruppenleitern und Hörern der Vorlesung für ihre Fragen, Anregungen und das Aufdecken von Fehlern.